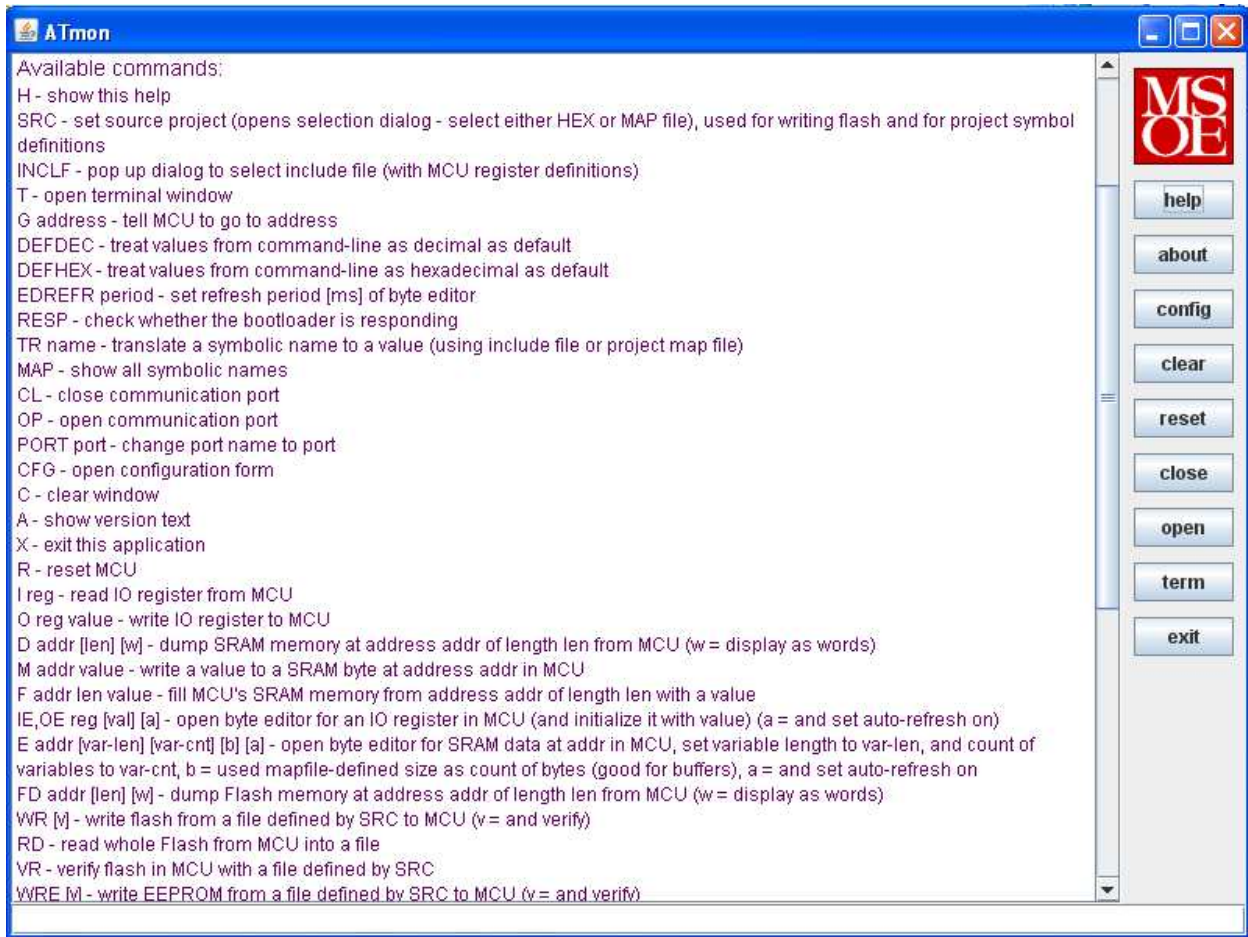


Using Atmon

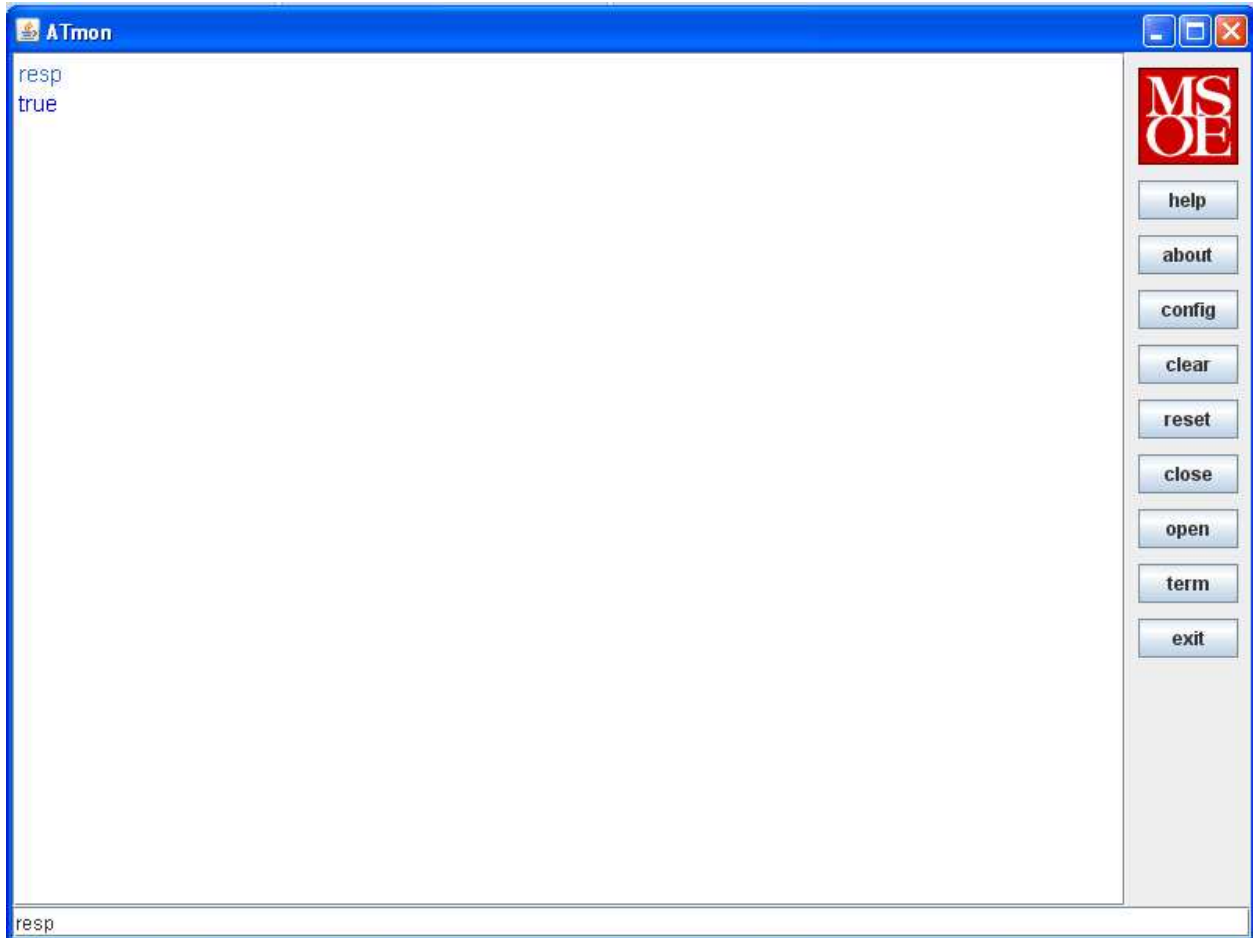
Copyright © 2007 William Barnekow <barnekow@msoe.edu>
All Rights Reserved

This guide is meant only to introduce you to the use of the Atmon bootloader/monitor program. It is not complete in that not all commands are discussed. Once you see how easy it is to use, you should experiment on your own to discover other useful commands.

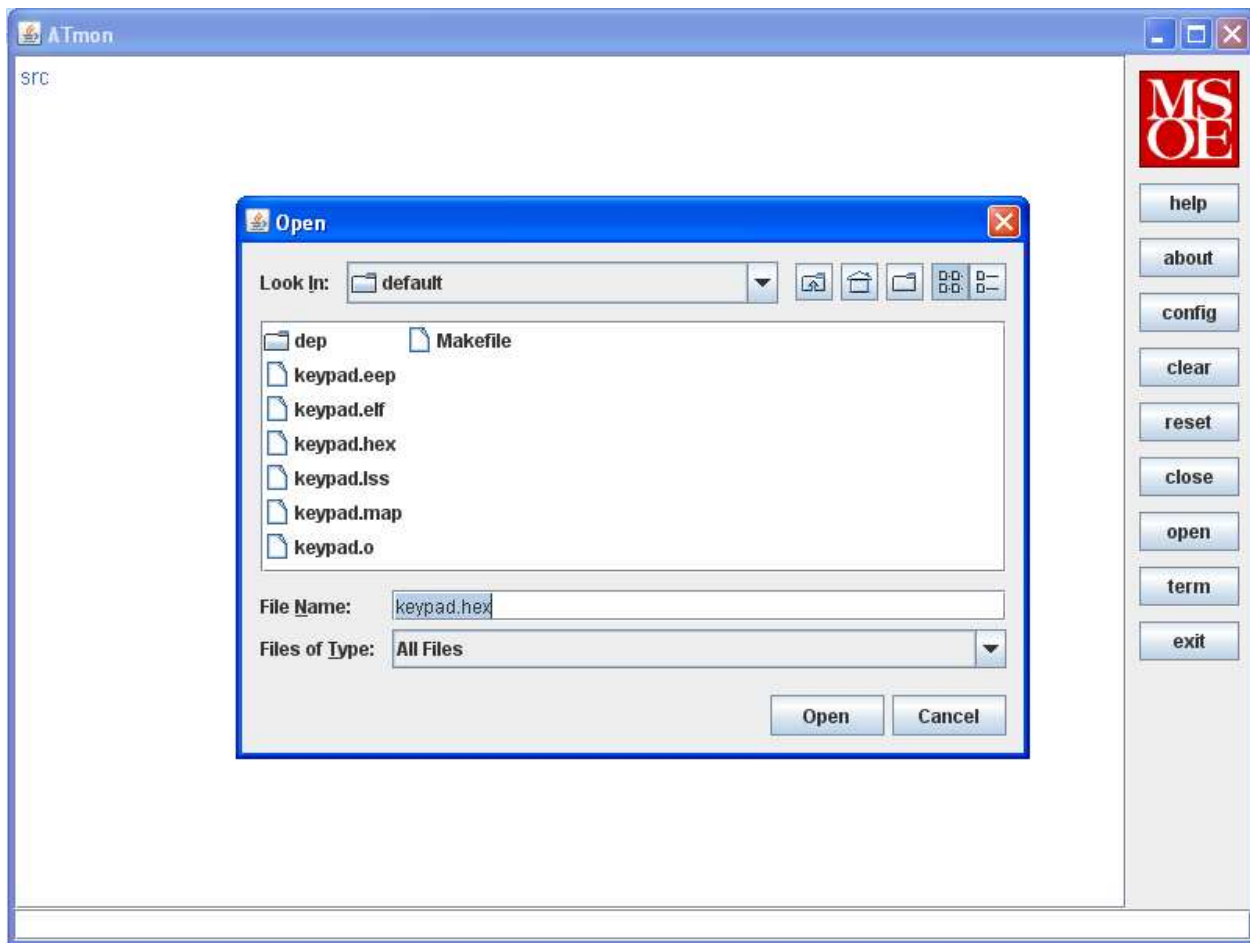
Atmon is a monitor/bootloader program for use with the Atmega32. It was written by Marek Smid, a visiting student at Milwaukee School of Engineering from the Czech Republic. The project was done under the direction of Professor William Barnekow. Using Atmon, it is possible to download a HEX file to any board that contains an Atmega32 and a serial communications port. The monitor is a JAR file that sends commands to the Atmega32 by way of the serial port. The commands are interpreted by the bootloader program that resides in the boot section of the Atmega32's flash memory. In addition to the download capability of Atmon, it also allows the user to monitor internal systems. Among the rich variety of commands are *memory display*, *memory write*, *register display*, *register write*, *I/O display* and *I/O write*. A complete list of all commands and a brief description of each can be obtained by clicking the help button (see below).



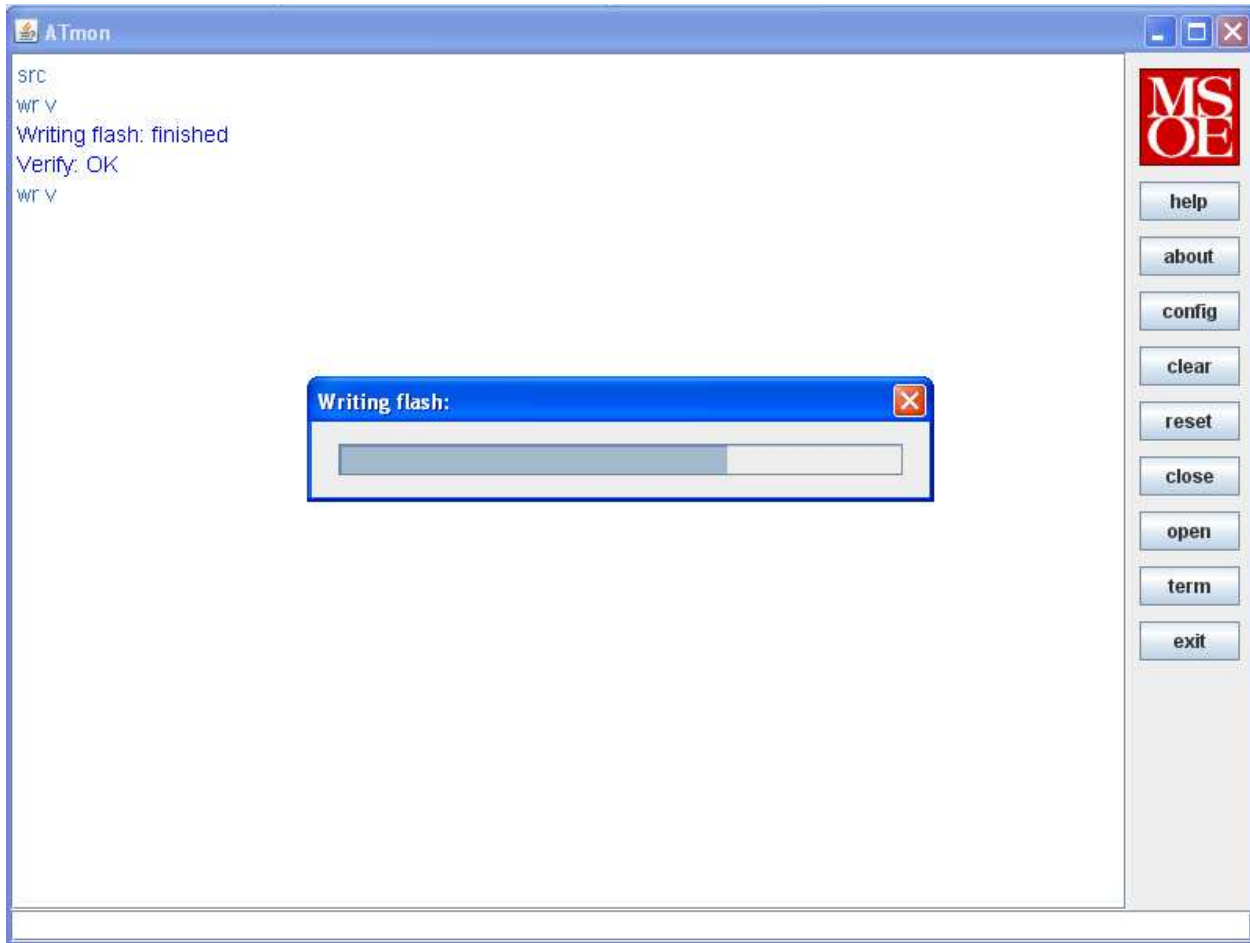
One very useful command is the *resp* command. This command allows one to determine if Atmon is communicating with the Atmega32. On the command line, enter *resp*. If a connection is valid, Atmon responds with *true*.



To download a HEX file, enter the command *src*. This will allow the user to browse to the desired file.



When the desired file is located click *open*. This needs to be done only once (unless a different file is to be downloaded). To download the program use the write and verify command, *wr v*. You will see a progress bar that informs you that the write is occurring. When finished writing and verifying, Atmon sends a message stating that writing is finished and verification is OK (see below).



When the process is complete, Atmon transfers control to the application and it begins to execute.

Important note: Atmon requires its own stack area. Applications that use the stack (all programs except the most trivial) must reserve 0x20 bytes of stack space for Atmon. This is done as follows:

```
ldi r16, hi8(RAMEND-0x20)
```

```
out SPH, r16
```

```
ldi r16, lo8(RAMEND-0x20)
```

```
out SPL, r16
```

It is a common mistake for beginning programmer's to not manage the stack properly. Common mistakes include

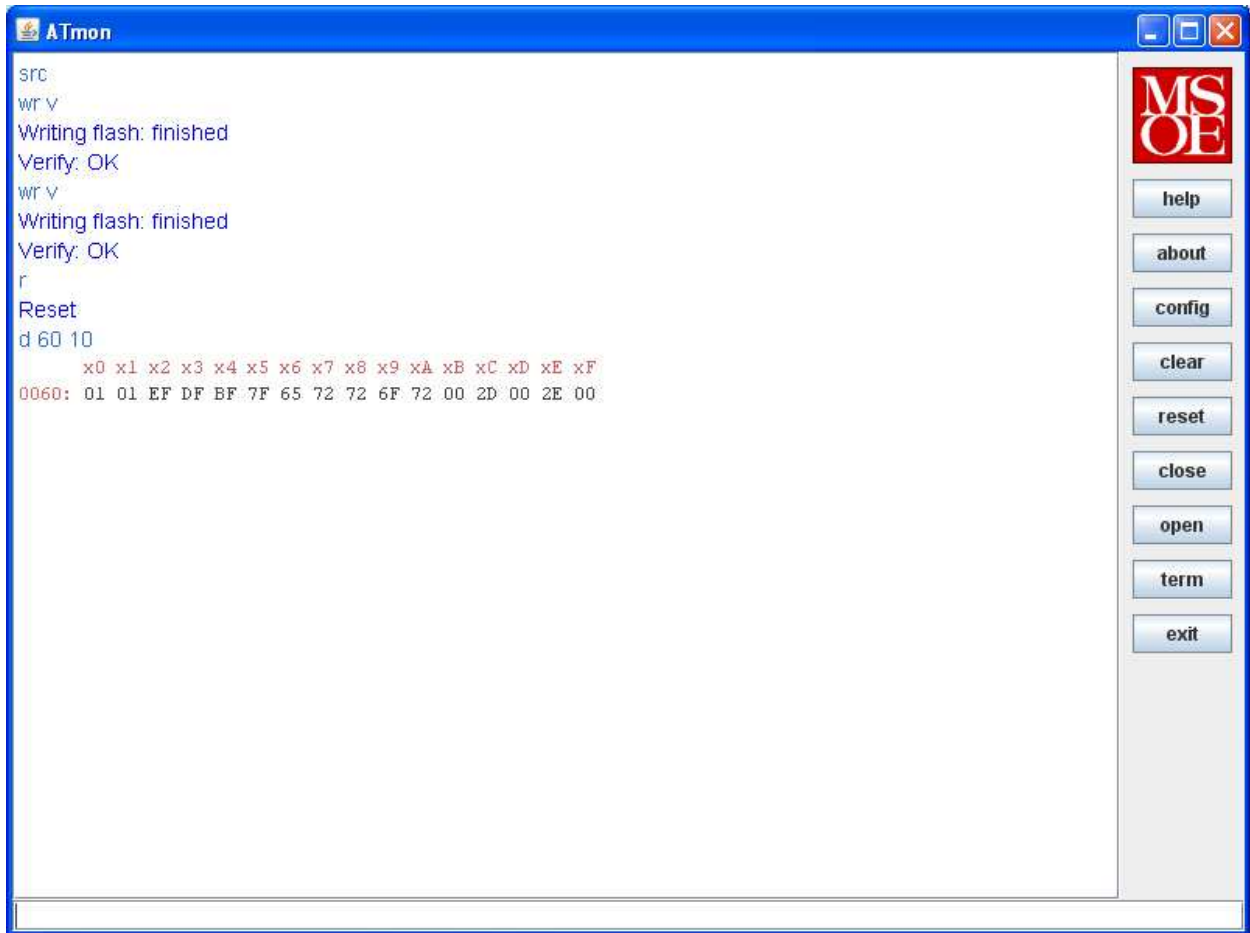
- Forgetting to initialize the stack pointer
- Jumping into a subroutine or interrupt routine

- Not popping every register that is pushed

This will cause the stack pointer to be misaligned resulting in improper execution of the bootloader/monitor program since it also uses the stack. If you use the *resp* command, Atmon will return *false*. To recover from this condition, Atmon has a stop command, *s*. To restore Atmon operation, cycle the power off/on or perform a reset. The stop command must be entered within **2 seconds**. This will prevent Atmon from transferring control to the application thereby not allowing the application to corrupt the stack. Make corrections to your program and download it again. This will work 90% of the time. However, occasionally, due to improper operation of Atmon it will change Atmega32 lock bits. This will usually require the lock bits to be changed using AVR studio and an In-System programmer. You will know if this is necessary because verification will fail when you attempt to download your program. The stop command can be used at any time to halt your program. To restart your program use the resume from stop command, *rs*.

Now, back to Atmon commands. A very useful command is display memory, *d address (length)*. This allows the user to examine the contents of SRAM locations as well as the programmable registers, r0 – r31.

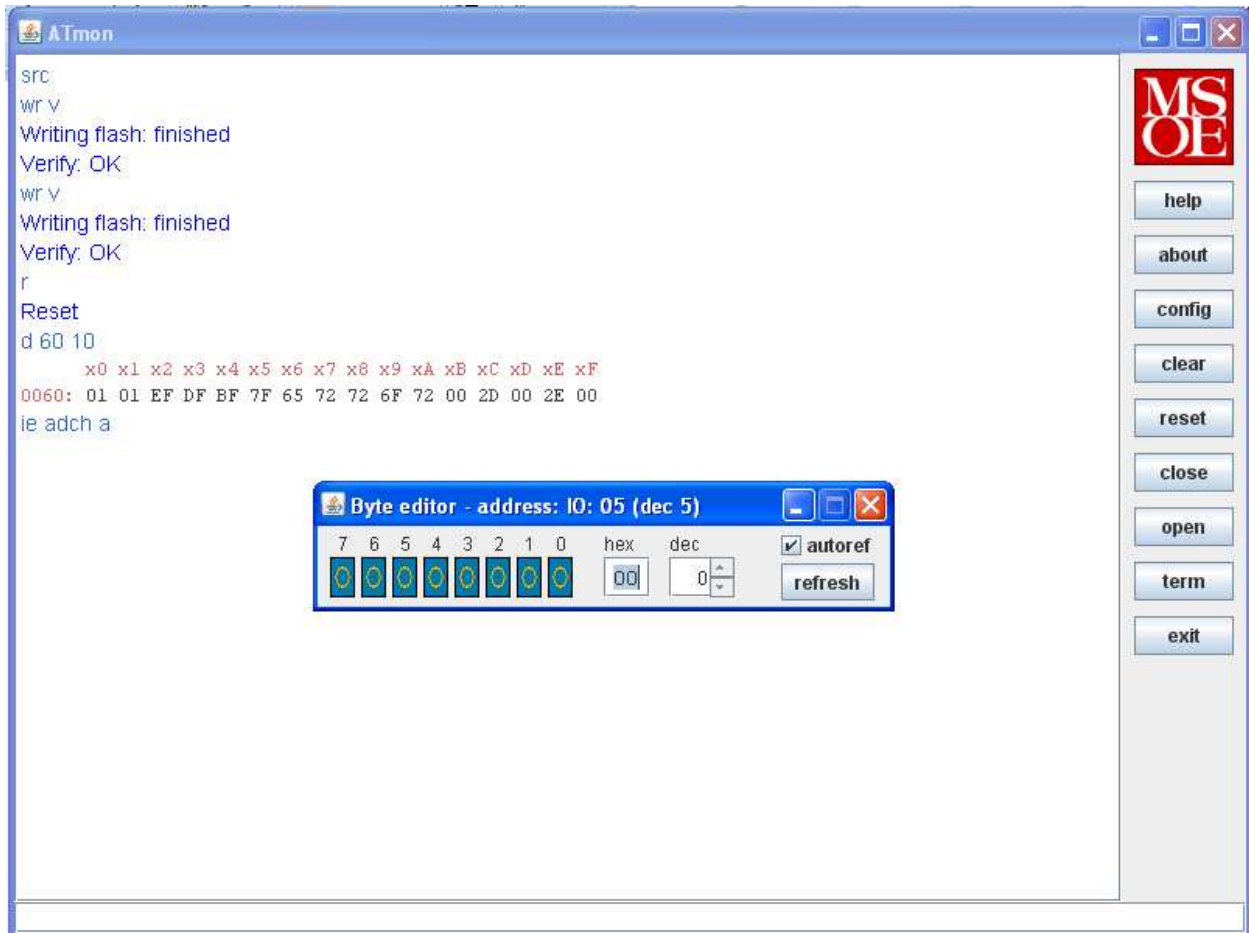
Below is the result of issuing the command *d 60 10*. This displays 0x10 locations in SRAM starting at location 0x60.



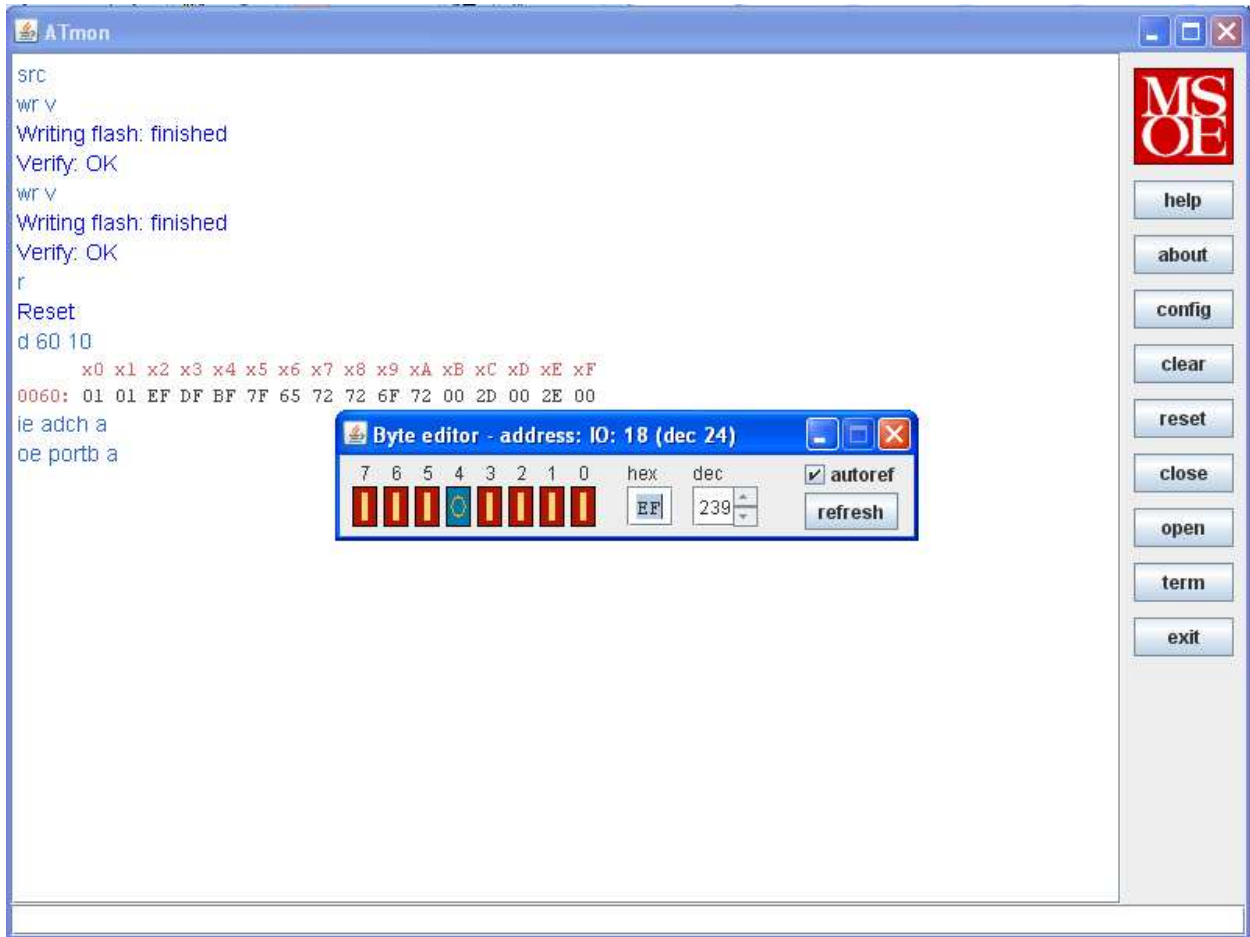
You may write to SRAM locations by using the memory modify command, *m location value*. For example, to write the value 0x55 to location use the command *m 60 55*. You can also read and modify the programmable registers using the display and modify commands. For example, to read r31 use the command *d r31*.

The contents of the I/O registers can also be read by using the input command, *i port*. For example to read PINA, use the command *i pina*. Atmon returns the value present on the PORTA pins. In a similar manner, you can write to the I/O registers by using the output command, *o port value*. For example, to write the value 0xAA to PORTC, use the command *o portc 0xaa*.

A way to watch the contents of an I/O port change in (almost) real time, you can use the edit command. For example, if the program is periodically writing to an output port, to monitor this use the output editor command, *oe port a*. This allows you to view the changes in the port with automatic refresh. For example, to watch the changes occurring in the ADCH register, use the command *ie adch a*. See below for an example.



There is also an output edit command. This command opens a byte editor window like the one used for input editor. The neat thing about this is that you can toggle individual bits by simply clicking the bit in the editor window. This changes the corresponding bit on the hardware output port. For example, to open the editor window on PORTB, use the command *oe portb a*. This will open an editor window as seen below.



There is also a byte editor command that allows you to view the registers and memory locations in (almost) real time. For example, to monitor the contents of r25, use the command `e r31 a`.

You should look at all the commands to see other operations that you may find useful. For example, if you told the assembler to create a listing file, you can view it in Atmon by using the list command, `lst`.

```

ATmon
161e: 90 95          com     r25
1620: bc 01        movw   r22, r24
1622: cd 01        movw   r24, r26
1624: 08 95          ret

00001626 <__divmodhi4>:
1626: 97 fb        bst    r25, 7
1628: 09 2e        mov    r0, r25
162a: 07 26        eor    r0, r23
162c: 0a d0        rcall  .+20      ; 0x1642
162e: 77 fd        sbrc   r23, 7
1630: 04 d0        rcall  .+8       ; 0x163a
1632: e5 df        rcall  .-54      ; 0x15fe
1634: 06 d0        rcall  .+12      ; 0x1642
1636: 00 20        and    r0, r0
1638: 1a f4        brpl   .+6       ; 0x1640

0000163a <__divmodhi4_neg2>:
163a: 70 95        com    r23
163c: 61 95        neg    r22
163e: 7f 4f        sbci   r23, 0xFF ; 255

00001640 <__divmodhi4_exit>:
1640: 08 95          ret

00001642 <__divmodhi4_neg1>:
1642: f6 f7        brtc   .-4       ; 0x1640
1644: 90 95        com    r25
1646: 81 95        neg    r24
1648: 9f 4f        sbci   r25, 0xFF ; 255
164a: 08 95          ret

```

There are many other interesting uses of Atmon. For example, you can test an I/O device without actually writing a program. You can set up the data direction registers by using the out command. You could even test an LCD interface by simply sending out the appropriate sequence of commands. Use your imagination and do not be afraid to try the commands.

Using the term command (t)

Atmon has a built-in terminal emulator that can be used to test application programs that use the USART (Universal Synchronous/Asynchronous Receiver/Transmitter). There is an inherent problem associated with using the USART while using Atmon at the same time, since Atmon also uses the USART for its serial communications. Once your program is downloaded, it immediately begins to run. Since the application takes over the USART, commands to Atmon can no longer be entered. To overcome the problem, you can use the stop command, *s*, to stop your application from executing. Just cycle the power or press the reset button on the development board. You then have two seconds to enter the stop command. At this point, Atmon still has control over the USART. Therefore, the terminal emulation command can be issued by

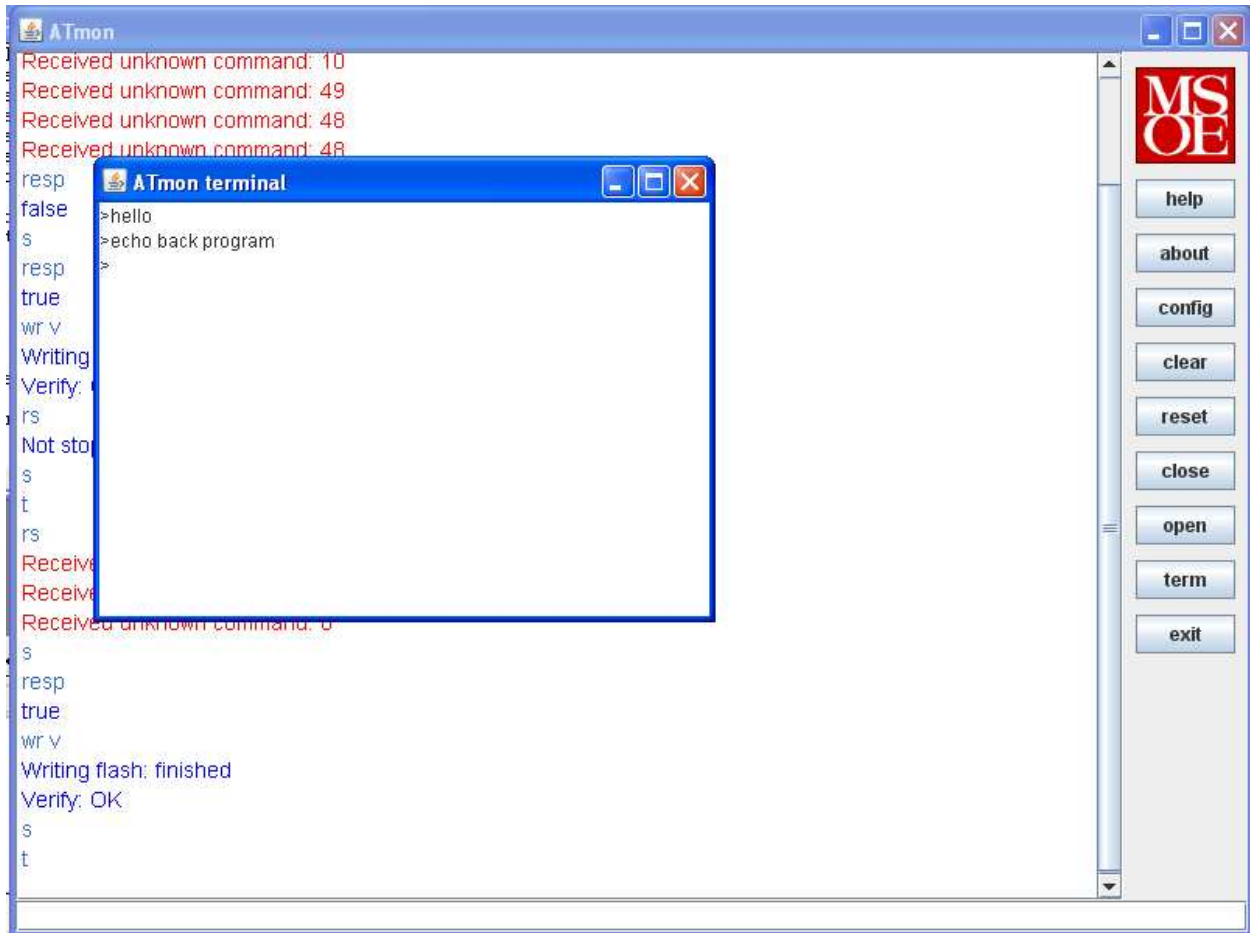
typing *t* on the command line. This turns the control of the USART over to the application. To start your application, either cycle the power or press the reset button. Here is a sample program that you can use to try out the terminal emulator.

```
.NOLIST
.include "m32def.inc"
.LIST
.equ F_CPU           = 8000000
.equ UART_BAUD_RATE = 19200
.equ UBRR_VAL       = (F_CPU/(UART_BAUD_RATE*16))-1
.equ CR             = 0x0d
.equ LF             = 0x0a
.org 0
    rjmp start
.org 0x2a
start:
    clr r16
    out UBRRH, r16
    ldi r16, UBRR_VAL
    out UBRRL, r16                ; Set baud rate to 19200
    ldi r16, (1<<RXEN)|(1<<TXEN) ; Enable Tx and Rx
    out UCSRB, r16
send_prompt:
    ldi r16, '>'
prompt:
    sbis UCSRA, UDRE
    rjmp prompt
    out UDR, r16
poll_rx:
    sbis UCSRA, RXC                ; Wait for character to be typed
    rjmp poll_rx
    in r16, UDR                    ; Get the character
poll_tx:
    sbis UCSRA, UDRE                ; Make sure transmitter is ready
    rjmp poll_tx
    out UDR, r16                    ; Echo back the character that was typed
    cpi r16, LF
    breq send_prompt
    cpi r16, CR
    breq send_lf
    rjmp poll_rx                    ; Repeat forever
send_lf:
```

```
ldi r16, LF
rjmp poll_tx

forever:
rjmp forever
```

To try it, just create an assembly language project and copy/paste the above program into the project. Then, build it as usual. After the program is assembled, it can then be downloaded using Atmon. Now, you will have to issue the stop command as described above. After entering the terminal emulation mode, just reset your board to run the application. All the program does is display a prompt. Any character you type should be displayed. When you hit the *enter* key, you should see the prompt displayed on the new line. All characters you type should be echoed back to the terminal window. See the screenshot below.



To leave the terminal emulation mode, just close the terminal window. Your application is still running, so it has to be stopped. First, use the resume from stop command, *rs*. Then cycle the power on your board or press the reset button and enter the stop command within two seconds. This returns control of the USART back to Atmon.